Table of Contents

Introduction

This page describes the preparation of tags and the validation of releases for Persistency Framework software.

The release process is described only for LCGCMT_64 or higher. In particular:

- we only describe CORAL and COOL releases here, while we no longer describe POOL (last supported in the LCGCMT_61 series);
- we only describe Linux and MacOSX, while we no longer describe Windows;
- we expect releases to be deployed under /cvmfs/sft.cern.ch/lcg/app/releases on CVMFS.

Note also that we expect CORAL and COOL to be checked out from SVN and no longer from CVS.

Persistency software release tag preparation

Production releases of CORAL and COOL are prepared whenever one of the experiments using this software (ATLAS, CMS, LHCb) require it. In the release process for ATLAS and LHCb, the software is built and tested with CMT by the SPI team using the nightly build infrastructure. The resulting libraries, binaries and scripts are then installed in the AFS release area under /afs/cern.ch/sw/lcg/app/releases, and later in CVMFS under /cvmfs/sft.cern.ch/lcg/app/releases. ATLAS and LHCb take the CORAL and COOL software already built from the AFS release area and integrate it into their build and runtime infrastructure using CMT. The process is different in CMS, where the CORAL source code is built internally using another configuration management tool, scram (this is the reason why scram BuildFile's can be found in CORAL).

Before launching the build of a new release (for ATLAS or LHCb using the SPI infrastructure, for CMS using its own infrastructure), the Persistency team members prepare new release tags for the two projects and communicate them to the SPI and experiment librarians.

If you are a member of the Persistency team and need to prepare release tags, please remember the following:

- Please document all changes in the new releases in `/afs/cern.ch/project/lcg/app/www/persist/coral/ReleaseNotesPF.txt`. This file on AFS is the release notes on this twiki, where it is displayed via php. Please also remember to commit the release notes to SVN from AFS.
- The release tag for version X.Y.Za of the software is PACKAGE_X_Y_Za, where: PACKAGE is one of CORAL or COOL; X, Y, Z are 1 to 2 digit numbers (a change in the major X signals API changes or relational schema changes that require changes in user code and/or schema evolution of the data; a change in the minor Y signals API extensions that break binary compatibility but only require a build of user code; a change in the patch Z signals ABI-compatible internal changes that do not require a rebuild of user code); 'a' is an optional letter indicating a rebuild of the same X.Y.Z user code against a different set of externals (note that a rebuild of user code may be required in some cases, e.g. if the ROOT dependency changed).
- For new CORAL releases (excluding rebuilds) the X.Y.Z release version is defined and hardcoded in CoralBase/CoralBase/VersionInfo.h: remember to *always* update this file!
- For new COOL releases (excluding rebuilds) the X.Y.Z release version is defined and hardcoded in CoolKernel/CoolKernel/VersionInfo.h: remember to *always* update this file!
- Every new production or development platform supported by CORAL must be explicitly declared in cmake variable platform_hashes in the CORAL CoralTest/CMakeLists.txt. This is needed to avoid cross-platform clashes between table names and/or port numbers in the nightly and release tests (see bug #102696).

Persistency software release validation

The release validation process is presently performed by the SPI team, at the same time as the release build, using this documentation provided by the Persistency team (see PF-1 "Automatize post-install procedure").

For the **SPI team: please remember to use the "sftnight" account**, other accounts (e.g. "lcgspi") may give rise to authentication or dblookup problems in CORAL.

The main part of the validation consists in running the COOL tests on all platforms: these tests internally cover most CORAL functionalities and also require a basic ROOT installation. In addition, the CORAL and CORAL server tests are also executed on a single platform: this is mainly to check the CORAL server functionalities more extensively. As a preliminary step, a few basic checks and/or configuration actions are also executed for all of CORAL, COOL and ROOT.

IMPORTANT WARNING!

In the following, the latest LCGCMT_67a release, including CORAL_2_4_1 and COOL_2_9_1 is taken as an example.

In the following, tag names specific to the LCGCMT_67a release are used. You may need to modify these commands within this twiki page for the specific new release you want to validate: please check before you blindly copy/paste them! Note anyway that in SVN, differently from CVS, logs are committed directly to their tag, so the risk of making mistakes is somewhat lower.

1. Preliminary checks and actions

Check availability of CORAL and COOL builds on AFS.

A simple `ls`

```
\ls -F /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1
\ls -F /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1
```

should yield:

```
i686-slc6-gcc47-opt/  logs/  x86_64-slc6-gcc48-dbg/
include/              src/   x86_64-slc6-gcc48-opt/
```

In other words, each release should include three shared directories (`include`, `logs`, `src`) and all relevant platform-specific directories (five in this case, all for Linux).

Check availability of ROOT builds on AFS.

Experience from several past releases shows that the release validation may fail (for instance in the PyCool tests) simply because the relevant ROOT release is not on AFS. A simple `ls`

```
\ls -F /afs/cern.ch/sw/lcg/app/releases/ROOT/5.34.13/
```

should normally yield:

```
i686-slc5-gcc47-dbg/        x86_64-mac108-gcc42-opt/  x86_64-slc6-gcc46-opt/
i686-slc5-gcc47-opt/        x86_64-slc5-gcc46-dbg/    x86_64-slc6-gcc47-dbg/
i686-slc6-gcc47-dbg/        x86_64-slc5-gcc46-opt/    x86_64-slc6-gcc47-opt/
i686-slc6-gcc47-opt/        x86_64-slc5-gcc47-dbg/    x86_64-slc6-gcc48-dbg/
```

```
src/                          x86_64-slc5-gcc47-opt/    x86_64-slc6-gcc48-opt/
x86_64-mac108-clang42-opt/  x86_64-slc6-gcc46-dbg/
```

In other words, the ROOT release should include the shared `src` directories and at least all platform-specific directories needed by COOL.

Check ACL of CORAL and COOL builds on AFS.

Several of the actions described below require write access (`rlidwka`) on the AFS installation. The best way to handle this is to grant/revoke yourself these privileges: to do that, however, you need administrator rights (`rlka`). Check with `fs la` if you have these rights at least on the top level directories (if you do, you probably have the same rights on the directories below, too).

```
fs la /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1
fs la /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1
```

This should return complex ACLs, including at least one of the following (or possibly both). You may also have more than `rlka`, e.g. `rlidwka`, but the bare minimum (and recommended set) is `rlka`.

```
lcgapp:coraladm rlka
lcgapp:cooladm rlka
```

Get write access privileges on the AFS installations

Several of the actions described below require write access to the AFS installation (e.g. on several log file directories, to rerun the test suites in situ). You may check using `fs la` if you have the required ACL's. In case of doubts or errors, grant yourself or ask someone to grant you AFS write privileges (across the full `src` trees of the three packages: this is not strictly required, but is just much easier). For instance, I generally grant myself `rlidwka` (while I have `rlka` as project administrator).

```
find /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src -type d -exec fs setacl -acl avalas
find /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1/src -type d -exec fs setacl -acl avalassi
```

Check consistency with latest SVN tags

As the tags are often updated at the last minute, it can be useful (though not strictly necessary) to cross check that the installed releases are in sync with SVN:

```
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src
svn diff --old=svn+ssh://svn.cern.ch/reps/lcgcoral/coral/tags/CORAL_2_4_1 --new=.
cd /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1/src
svn diff --old=svn+ssh://svn.cern.ch/reps/lcgcool/cool/tags/COOL_2_9_1 --new=.
```

These should return no differences.

Create (or recreate) CMT setup and cleanup scripts (cmt config)

In the following it is assumed that the `tcsh` shell is used on an SLC6 system like `lxplus6`. Debug 64bit builds with the production compiler are generally used as the default platform, for this step and many of the following lower down in this page. Execute the following commands to (re)create the CMT setup and cleanup (`.csh`, `.sh`) scripts, for CORAL and COOL (you probably need to source `CMT_env.csh` only once, but here this is done in each package independently for clarity).

```
setenv CMTCONFIG x86_64-slc6-gcc48-dbg
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src/config/cmt
source CMT_env.csh
```

```
cmt broadcast \rm -f setup.bat setup.csh setup.sh cleanup.bat cleanup.csh cleanup.sh
cmt broadcast cmt config
cmt broadcast ls -l '*up.*sh'
cd /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1/src/config/cmt
source CMT_env.csh
cmt broadcast \rm -f setup.bat setup.csh setup.sh cleanup.bat cleanup.csh cleanup.sh
cmt broadcast cmt config
cmt broadcast ls -l '*up.*sh'
```

A few observations:

- *NB 1: Do rerun `cmt config` even if you see that the setup and cleanup scripts are already there, because the existing scripts may have been created on the `/build/` directories, in which case they would point to the wrong installation!*
- *NB 2: Do remove any existing scripts before rerunning `cmt config`, because otherwise it may happen that the existing scripts are not overwritten! Use `\rm -f` because `\rm` (without `-f`) will fail with an error if any setup file is not there.*
- *NB 3: Do not forget to check with `ls -l` that the scripts have been (re)created, because you may miss the appropriate AFS privileges to write to the directories (and `cmt config` would fail silently in that case).*

2. Execute the COOL and CORAL tests

The COOL and CORAL tests are executed directly in the AFS release area, where the log files are also created. You therefore need write access to all relevant AFS directories, which you should have by now if you followed the instructions above. In addition, you also need SVN write access privileges to commit the log files back to the appropriate tag in the SVN repository. The releases are installed on AFS via svn+ssh checkout from SVN, which automatically allows writers to commit back to the repository if they have the relevant privileges.

2a. Execute the COOL tests

The main part of the validation consists in running the COOL tests on all platforms:

- On linux, they are executed on an SLC6 system like `lxplus5` and an SLC6 system like `lxplus6`, using the `tcsh` shell.
- On OSX, they are executed on `macphsft06`, using the `tcsh` shell.

Prepare database credentials for the COOL tests

All of the COOL and CORAL tests take their credentials for accessing databases from the CORAL `authentication.xml` and `dblookup.xml` files. While for CORAL and CORAL_SERVER tests the paths to these files are hardcoded for all users to be the same as those used by the `lcgnight` user that runs the nightlies (see PersistencyTests#DatabaseConfiguration), for the COOL tests a different convention is used:

- for users other than `lcgnight`, it is recommended to manually set the environment variables CORAL_AUTH_PATH and CORAL_DBLOOKUP_PATH before executing the COOL tests;
- the six aliases `COOL-(Oracle|MySQL|CoralServer-Oracle|CoralServer-MySQL|Frontier|FrontierCache)-$USER` must all be defined in `$CORAL_DBLOOKUP_PATH/dblookup.xml`.

For all users other than `lcgnight` I would then recommend to try the following:

- Set the authentication path to the default location for the nightlies tests, `setenv CORAL_AUTH_PATH /afs/cern.ch/sw/lcg/app/pool/db`

- Set the dblookup path to a location of your choice, for instance `setenv CORAL_DBLOOKUP_PATH $HOME/private/cool`
- Modify and install the default dblookup file in your dblookup path, e.g. `cat /afs/cern.ch/sw/lcg/app/pool/db/dblookup.xml | sed -r "s/COOL-(.*)-lcgnight/COOL-\1-$USER/" > $HOME/private/cool/dblookup.xml`
- Remember to also set these paths on Windows and copy your own `dblookup.path` if it is not accessible on Windows via AFS.

Execute the COOL tests

In practice, open as many `lxplus5`, `lxplus6` and `macphsft06` windows on your terminal as the number of platforms supported on each O/S. Make sure that you are using the `tcsh` shell and that you have an AFS token (on `lxplus5` and `lxplus6` you get it automatically via `ssh` at logon, but on `macphsft06` you need to explicitly `kinit` after logon). On each window, execute the following commands, where `<platform>` is the relevant platform you want to test on that window.

```
cd /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1/src/config/cmt
setenv CMTCONFIG <platform>
source CMT_env.csh
setenv LCG_NGT_SLT_NAME release
cd ../qmtest
./execQmtest.sh &
```

Note that the `setenv LCG_NGT_SLT_NAME release` command is a temporary hack needed by the CORAL server tests within COOL (to choose between the 'production' and 'development' version of the CORAL server protocol). If this step is omitted, the `pycoolutilities.coral.oracle.regression` test may fail with a "Wrong CAL version" error.

Create and check the qmtest summaries for the COOL tests

Once the tests are completed, use a qmtest wrapper `summarizeAll.csh` to create `<platform>.xml` and `<platform>.summary` reports from the different `<platform>.qmr` result files. You may do this on a single linux machine (this will be called the 'SVN' window in the following). Using `tail` and `svn diff`, visually inspect the `<platform>.summary` reports for any problem (do not check the `<platform>.xml` reports, as these are much more detailed and many differences are expected from `svn diff` even if all tests are successful).

```
cd /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1/src/config/cmt
setenv CMTCONFIG x86_64-slc6-gcc48-dbg
source CMT_env.csh
source setup.csh
cd /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1/src/logs/qmtest
./summarizeAll.csh all
tail *.summary
svn diff *.summary
```

If everything is correct, commit the files back to SVN for future reference. Also remove the .qmr files to avoid any interference with future activities.

```
cd /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1/src/logs/qmtest
svn commit -m "All OK for COOL_2_9_1 in LCGCMT_67a" *.xml *.summary
\rm -rf *.qmr
```

Note that logs are automatically committed to the relevant tag in SVN (while they are not committed to the trunk). Note that this is different from CVS, where files were committed to the HEAD of the MAIN branch and they had to be tagged explicitly a posteriori.

Rerun a single test in case of failure

If the previous step failed because of major issues (e.g. CORAL installation is missing or incomplete), rerun all tests after fixing the problems. If however only one test failed, you may try to rerun that single test immediately as the failure may be due to a temporary issue (e.g. an overload of the test database). You may also choose this approach if a well identified set of tests failed (e.g. PyCOOL tests because ROOT was improperly installed, MySQL tests because the MySQL server was down), and rerun only the relevant set of tests (a few qmtest sub-suites are defined to that purpose, you may for instance `./execQmtest.sh mysql` to execute only the MySQL tests).

Suppose for instance that only the pycoolutilities.frontier.regression_cache test fails on x86_64-slc6-icc13-dbg. That single test can then be rerun from the existing window where that platform had already been configured.

```
cd /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1/src/config/qmtest
./execQmtest.sh pycoolutilities.frontier.regression_cache &
```

Once the test is completed, create new reports (on the 'SVN' window) and check the result summary using `tail`.

```
cd /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1/src/logs/qmtest
./summarizeAll.csh all
tail *.summary
```

If the test was successful, use the `fullMergeTestOutput.csh` tool (still on the 'SVN' window) to merge the xml/summary reports for this subset of the test suite into the full reports (now committed in SVN), where this test was failing. This internally moves the partial logs and reports into a directory `new`, recovers the full reports via an `svn update`, runs the tool, copies the merged xml from the `new.merged` directory and recreates the report summaries from the xml files. Before committing back to SVN, check with `tail` that all tests pass as expected in the merged summaries. Clean up at the end.

```
cd /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1/src/logs/qmtest
./fullMergeTestOutput.csh *qmr
tail *.summary
svn commit -m "All ok for COOL_2_9_1 in LCGCMT_67a after rerunning pycoolutilities.frontier.reg
\rm -rf new* *pyc
```

2b. Execute the CORAL tests

The second part of the validation consists in running the CORAL tests. This is generally done on a single platform, as it is mainly to check the CORAL functionalities more extensively and it largely overlaps with the COOL release validation and the CORAL nightly tests. One additional reason to run the CORAL tests only on a single platform is that it is not yet fully verifed that there are no interferences (e.g. accesses to database tables with the same names) from tests running simultaneously on different platforms. The procedures and tools to run the tests, produce and handle the reports and any errors are exactly the same as previously described for COOL. The only difference for CORAL is the additional suggestion to execute the `createconfig.py` script (and then launching a `svn diff`) before executing the tests: this is just a cross-check that the qmtest test suite (i.e. the `.qms` directories) reflect the master list of CORAL tests in the `testlist.xml` file (if this is not the case, please report it).

Note that one of the CORAL tests (the pycoral.networkglitch test) internally executes an `ssh` to the local host, assuming that you properly configured your authentication settings. Before running the CORAL tests, make sure that you can execute: :

```
ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no $HOST
```

Now execute the tests: :

```
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src/config/cmt
setenv CMTCONFIG x86_64-slc6-gcc48-dbg
source CMT_env.csh
setenv LCG_NGT_SLT_NAME release
setenv LCG_NGT_SLT_NUM 0
cd ../qmtest
./createconfig.py -r
svn diff
./execQmtest.sh &
```

Note that the `setenv LCG_NGT_SLT_NUM 0` command is a temporary hack needed by the CORAL network glitch tests (to disable some tests that are known to fail, see bug #92391).

Produce and check the test reports (you can use the same window for CORAL, while for COOL it was good to have a separate window from all those needed to run tests on different platforms):

```
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src/config/cmt
source setup.csh
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src/logs/qmtest
./summarizeAll.csh all
tail *.summary
svn diff *.summary
```

If everything is as expected, commit the files back to SVN for future reference. Also remove the .qmr files to avoid any interference with future activities.

```
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src/logs/qmtest
svn commit -m "All as expected (only pycoral.networkglitch fails) for CORAL_2_4_1 in LCGCMT_67a
\rm -rf *.qmr
```

Note again that logs are automatically committed to the relevant tag in SVN (while they are not committed to the trunk). Note that this is different from CVS, where files were committed to the HEAD of the MAIN branch and they had to be tagged explicitly a posteriori.

If anything went wrong in a single test, you can rerun it as described above for COOL (the same exact tools exist in the CORAL and COOL directories).

2c. Execute the CORAL_SERVER tests

The third part of the validation consists in running the CORAL_SERVER tests. This is done on a single platform (possibly the production platform for the ATLAS HLT system, presently the only user of this software).

These tests are different from the COOL and CORAL test suites described above.

- To start with, the CORAL_SERVER tests are launched through a set of custom scripts that are not based internally on qmtest (unlike the COOL and CORAL tests).
- In addition, several CORAL_SERVER tests internally launch more than one process, as they may spawn a CoralServer (and in some cases also a CoralServerProxy) instance against which the tests are executed, and which are only started for the duration of the tests. For comparison, some of the qmtest-controlled tests in the COOL and CORAL test suites also test the CoralServer software, but they do so by executing tests against a well defined server that is up all the time.

The CORAL_SERVER test suite includes three sets of tests:

- the CoralAccess test set, executing a set of simple queries executed via PyCoral;

- the CoolRegression test set, executing the "regression" test from the COOL test suite;
- the HLT test set (the longest of the three), executing a standalone configuration of an ATLAS HLT process.

Each set of tests is internally executed in more than one of the following configurations.

- "local" mode: direct access to an Oracle database (2-tier: client, DB server)
- "Fac" mode: access to Oracle after encoding and decoding queries via the 'facade' component of the CORAL_SERVER software (2-tier: client, DB server)
- "StbFac" mode: access to Oracle after encoding and decoding queries via the 'stub' and 'facade' components of the CORAL_SERVER software (2-tier: client, DB server)
- "server" mode: access to Oracle via a CoralServer instance (3-tier: client, CORAL server, DB server)
- "proxy0" mode: access to Oracle via a CoralServer and a CoralServerProxy instances, with caching disabled in the proxy (4-tier: client, CORAL server, DB server)
- "proxy" mode: access to Oracle via a CoralServer and a CoralServerProxy instances, with caching fully enabled in the proxy (4-tier: client, CORAL server, DB server)
- "fontier" mode: for comparison, access to Oracle via a Frontier server

A single wrapper script needs to be executed to launch each of the three sets of tests. Internally, the wrapper takes care of executing each test for more than one configuration and of spawning any relevant CoralServer and/or CoralServerProxy instance, if needed. Success or failure for each test set can be immediately determined from the messages that are printed out on the screen at the end of the execution (they should all start with `OK`).

CoralAccess tests

To execute the tests (after setting up the environment as shown above):

```
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src/Tests/CORAL_SERVER/CoralAccess
./runThisTestSuite.sh file cwd &
```

The tests succeed if all lines start with 'OK'. When this was last checked, the tests would normally return:

```
OK: match OK (20) in clientLog_Fac.txt
OK: no 'error' in clientLog_Fac.txt
OK: match OK (20) in clientLog_local.txt
OK: no 'error' in clientLog_local.txt
OK: match OK (20) in clientLog_proxy0.txt
OK: no 'error' in clientLog_proxy0.txt
OK: match OK (20) in clientLog_proxy.txt
OK: no 'error' in clientLog_proxy.txt
OK: match OK (20) in clientLog_server.txt
OK: no 'error' in clientLog_server.txt
OK: match OK (20) in clientLog_StbFac.txt
OK: no 'error' in clientLog_StbFac.txt
OK: no 'terminated by signal' in client, server or proxy logs
OK: no 'segmentation violation' in client, server or proxy logs
OK: no Frontier logs to analyse
OK: no 'UNKNOWN' in client or server csv files
```

If everything is as expected, commit the files back to the SVN tag for future reference.

```
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src/logs/CORAL_SERVER/CoralAccess
svn commit -m "All as expected for CORAL_2_4_1 in LCGCMT_67a"
```

CoolRegression tests

To execute the tests (after setting up the environment as shown above):

```
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src/Tests/CORAL_SERVER/CoolRegression
./runThisTestSuite.sh file cwd &
```

The tests succeed if all lines start with 'OK'. When this was last checked, the tests would normally return:

```
OK: match OK (1) in coolClientLog_Fac.txt
OK: no 'error' in coolClientLog_Fac.txt
OK: match OK (1) in coolClientLog_local.txt
OK: no 'error' in coolClientLog_local.txt
OK: match OK (1) in coolClientLog_proxy0.txt
OK: no 'error' in coolClientLog_proxy0.txt
OK: match OK (1) in coolClientLog_proxy.txt
OK: no 'error' in coolClientLog_proxy.txt
OK: match OK (1) in coolClientLog_server.txt
OK: no 'error' in coolClientLog_server.txt
OK: match OK (1) in coolClientLog_StbFac.txt
OK: no 'error' in coolClientLog_StbFac.txt
OK: no 'terminated by signal' in client, server or proxy logs
OK: no 'segmentation violation' in client, server or proxy logs
OK: no Frontier logs to analyse
OK: no 'UNKNOWN' in client or server csv files
```

If everything is as expected, commit the files back to the SVN tag them for future reference.

```
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src/logs/CORAL_SERVER/CoolRegression
svn commit -m "All as expected for CORAL_2_4_1 in LCGCMT_67a"
```

HLT tests

NB For the moment, the HLT tests should only be executed for validating new releases in the LCG61 series; the tests cannot be executed for LCGCMT_67a because no ATLAS HLT release using the same Boost version as in LCGCMT_67a exists (bug #90431). The CoralAccess and CoolRegression tests instead should be executed for all release series, including LCGCMT_67a.

To execute the tests (after setting up the environment as shown above), call the wrapper script giving the current release as argument.

```
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src/Tests/CORAL_SERVER/HLT
./runThisTestSuite.sh file LCGCMT_67a &
```

The tests succeed if all lines start with 'OK'. When this was last checked, the tests would normally return:

```
OK: match OK (37) in hltClientLog_Fac.txt
OK: no issue in hltClientLog_Fac.summary
OK: match OK (37) in hltClientLog_frontier.txt
OK: no issue in hltClientLog_frontier.summary
OK: match OK (37) in hltClientLog_local.txt
OK: no issue in hltClientLog_local.summary
OK: match OK (37) in hltClientLog_proxy0.txt
OK: no issue in hltClientLog_proxy0.summary
OK: match OK (37) in hltClientLog_proxy.txt
OK: no issue in hltClientLog_proxy.summary
OK: match OK (37) in hltClientLog_server.txt
OK: no issue in hltClientLog_server.summary
OK: match OK (37) in hltClientLog_StbFac.txt
OK: no issue in hltClientLog_StbFac.summary
OK: no 'terminated by signal' in client, server or proxy logs
OK: no 'segmentation violation' in client, server or proxy logs
```

```
OK: no 'error' in Frontier logs
OK: no 'UNKNOWN' in client or server csv files
```

If everything is as expected, commit the files back to the SVN tag for future reference.

```
cd /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src/logs/CORAL_SERVER/HLT
svn commit -m "All as expected for CORAL_2_4_1 in LCGCMT_67a"
```

3. Final checks and actions

Get rid of write access privileges on the AFS installations

Finally, revoke from yourself write access to the AFS installation. For instance, I generally grant myself `rlka` (I had granted myself `rlidwka` at the beginning).

```
find /afs/cern.ch/sw/lcg/app/releases/CORAL/CORAL_2_4_1/src -type d -exec fs setacl -acl avalas
find /afs/cern.ch/sw/lcg/app/releases/COOL/COOL_2_9_1/src -type d -exec fs setacl -acl avalassi
```

-- AndreaValassi - 17-Aug-2011

This topic: Persistency > PersistencyReleaseProcess
Topic revision: r63 - 2016-06-24 - AndreaValassi